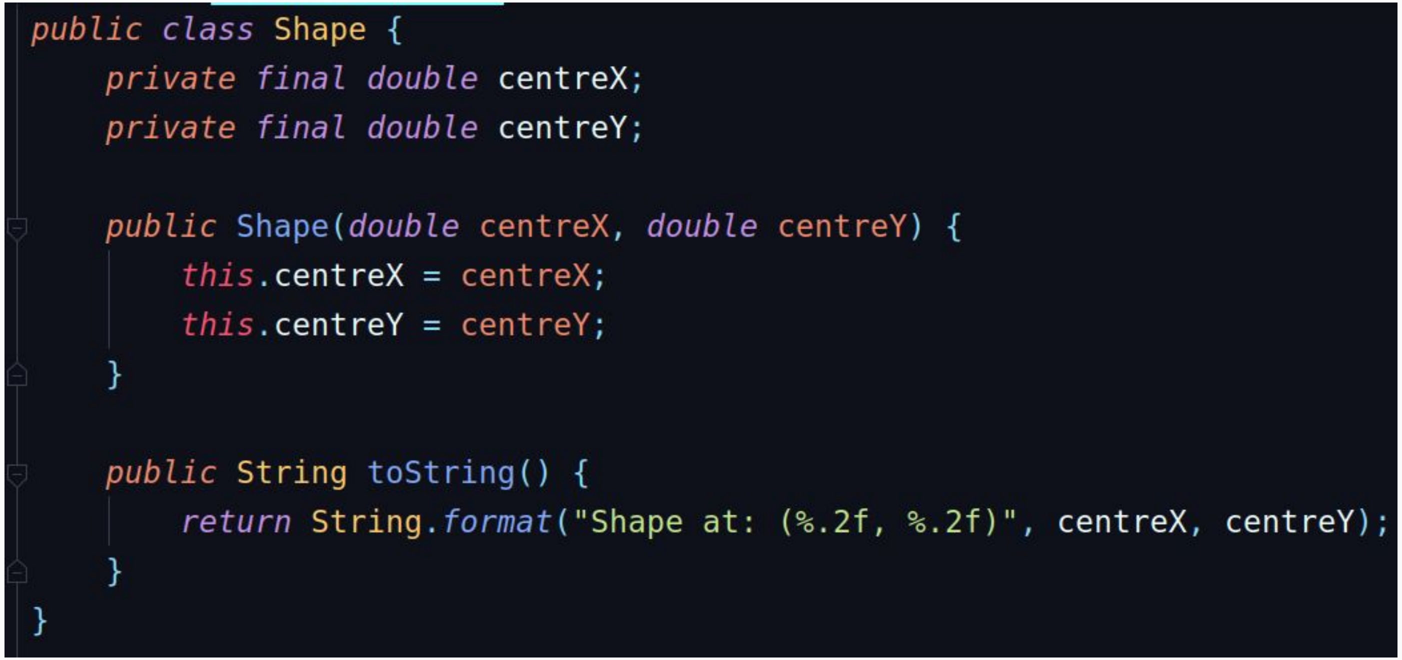
SWEN20003

Workshop 5, Week 6

Eleanor McMurtry, University of Melbourne

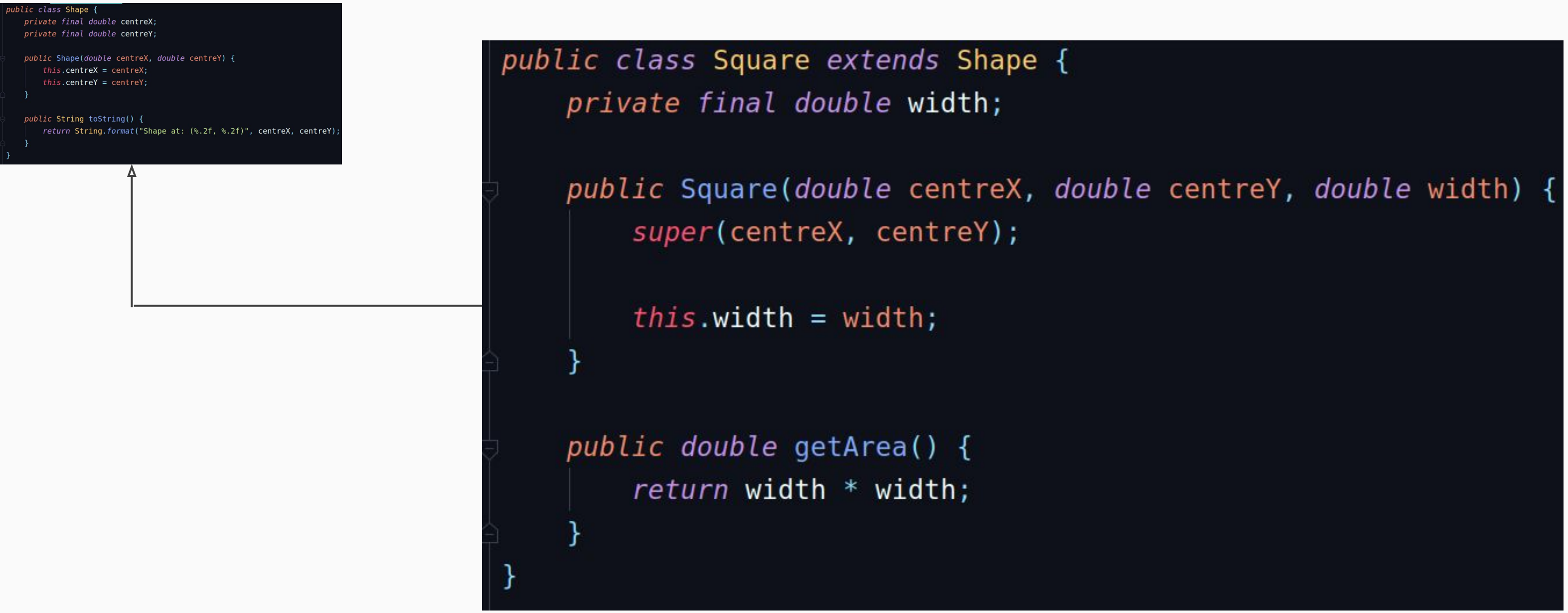
Inheritance

Building on top of existing classes



```java
public class Shape {
    private final double centreX;
    private final double centreY;

    public Shape(double centreX, double centreY) {
        this.centreX = centreX;
        this.centreY = centreY;
    }

    public String toString() {
        return String.format("Shape at: (%.2f, %.2f)", centreX, centreY);
    }
}
```

Building on top of existing classes



```java
public class Shape {
    private final double centreX;
    private final double centreY;

    public Shape(double centreX, double centreY) {
        this.centreX = centreX;
        this.centreY = centreY;
    }

    public String toString() {
        return String.format("Shape at: (%.2f, %.2f)", centreX, centreY);
    }
}
```

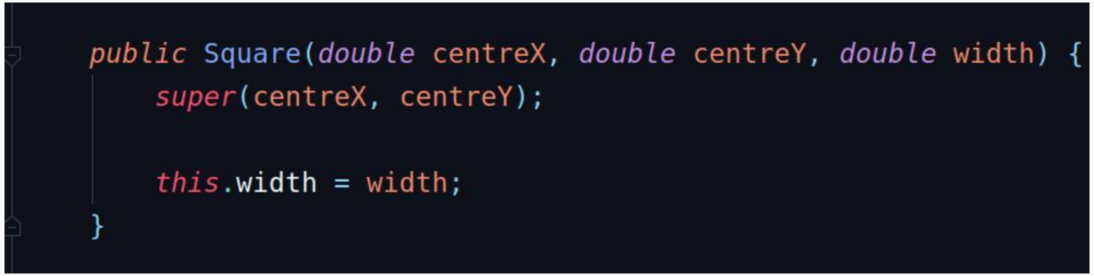
```java
public class Square extends Shape {
    private final double width;

    public Square(double centreX, double centreY, double width) {
        super(centreX, centreY);

        this.width = width;
    }

    public double getArea() {
        return width * width;
    }
}
```

Building on top of existing classes

- `Square` **inherits** the `centreX` and `centreY` attributes, and the `toString()` method.
- It **has another method**, `getArea()`.

The super keyword

- Shape does not have a **default constructor**. We can call its constructor using `super`.

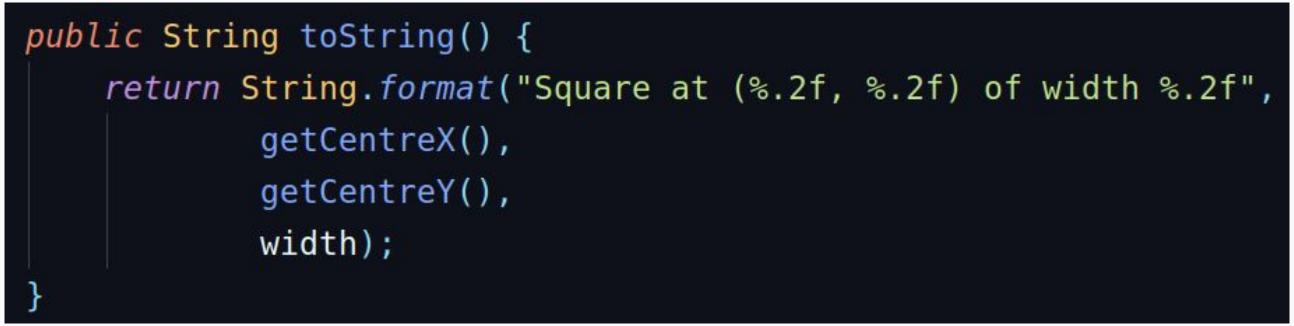


```
public Square(double centreX, double centreY, double width) {
    super(centreX, centreY);

    this.width = width;
}
```

- We could also access any public attributes or methods of Shape using `super`, similarly to `this`.

Method overriding

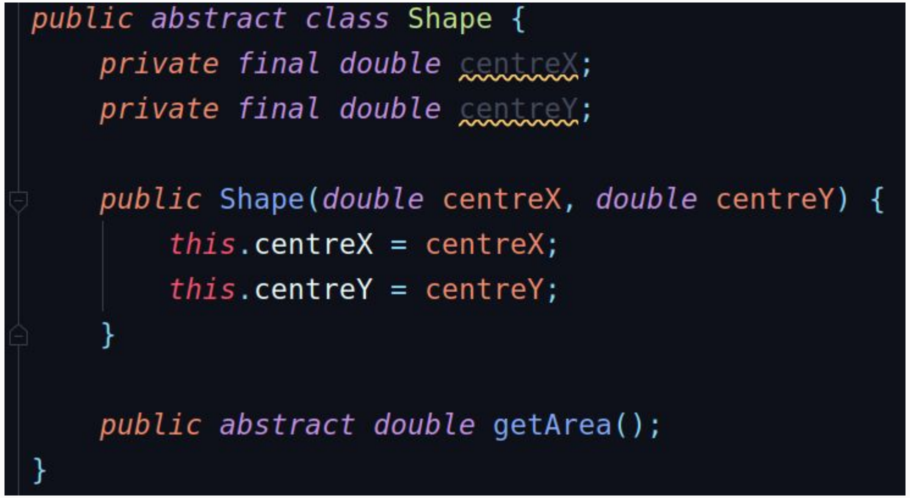
- We can **override** superclass methods by defining another method with the **same name and arguments**.



```java
public String toString() {
    return String.format("Square at (%.2f, %.2f) of width %.2f",
            getCentreX(),
            getCentreY(),
            width);
}
```

Abstract classes

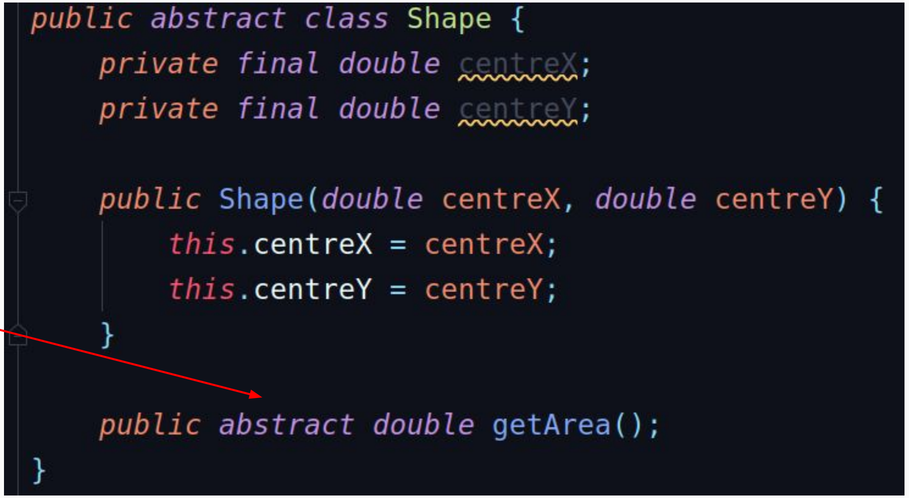
- A class that is not completely defined
- Cannot create any instances



```java
public abstract class Shape {
    private final double centreX;
    private final double centreY;

    public Shape(double centreX, double centreY) {
        this.centreX = centreX;
        this.centreY = centreY;
    }

    public abstract double getArea();
}
```

Abstract classes

- A class that is not completely defined
- Cannot create any instances



```java
public abstract class Shape {
    private final double centreX;
    private final double centreY;

    public Shape(double centreX, double centreY) {
        this.centreX = centreX;
        this.centreY = centreY;
    }

    public abstract double getArea();
}
```

abstract method:
definition provided by subclasses

Polymorphism

- Objects/methods may have different meanings in different contexts

Polymorphism

- Objects/methods may have different meanings in different contexts
  - Overloading methods

Polymorphism

- Objects/methods may have different meanings in different contexts
  - Overloading methods
  - Overriding methods

Polymorphism

- Objects/methods may have different meanings in different contexts
  - Overloading methods
  - Overriding methods
  - Substitution

Polymorphism

- Objects/methods may have different meanings in different contexts
  - Overloading methods
  - Overriding methods
  - Substitution
  - Generics

Polymorphism

- Objects/methods may have different meanings in different contexts
  - Overloading methods
  - Overriding methods
  - **Substitution**
  - Generics